Report Lab 2

Search Engines DD2424

In this assignment, mini-batch gradient descent was used to classify images from CIFAR-10 into 10 classes. The network had two layers and L2 regularization was used. The training set was of size 10 000 and had 3072 features. The hidden layer of the network had 50 nodes.

**Gradients**

The gradients of W and b were calculated in the backward pass. These gradients were checked for correctness against gradients computed with the central difference formula.

If the result of the following two equations held, gradients were considered as being correct.

(1)

(2)

Where is the analytically computed gradient, is the numerically computed gradient and

.

Table 1. Resulting difference between analytical and numerical

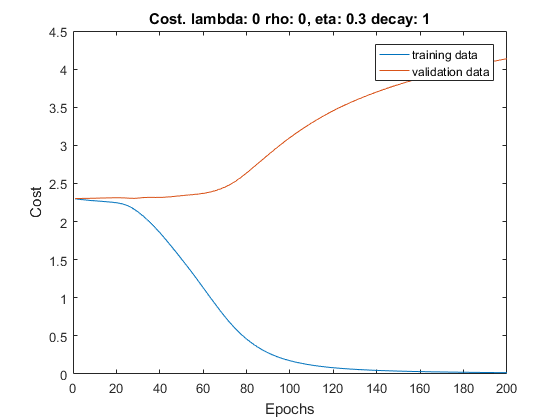
gradients, using only 10 input samples with 700 dimensions.

|  |  |  |
| --- | --- | --- |
|  | Equation 1, Max difference | Equation 2, Relative error |
| W1 | 6.561e-11 | 2.715e-08 |
| W2 | 5.3503e-11 | 3.632e-06 |
| b1 | 4.1344e-11 | 5.8416e-08 |
| b2 | 2.0409e-11 | 3.0724e-07 |

When looking at table 1, you can draw the conclusion that the computed gradients are correct.

**Momentum**

Momentum and learning rate decay were implemented. As you can see from comparing figure 1 to figure 2, training was drastically speeded up when using momentum. With momentum and decay, the network reached a cost below 0.25 in about 50 epochs. The same result took about 90 epochs without momentum. In figure 3 you can see that when momentum but no decay is used, the training can be speeded up further. A zero cost was reached in only 30 epochs, but it can be noted that the network was overfitted to great extent.



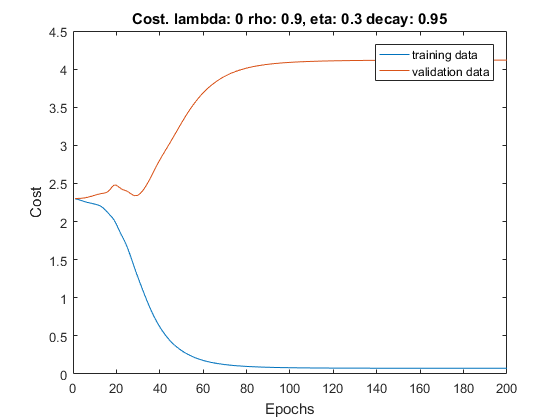


Figure 1. Overtraining without momentum Figure 2. Overtraining with momentum and

decay

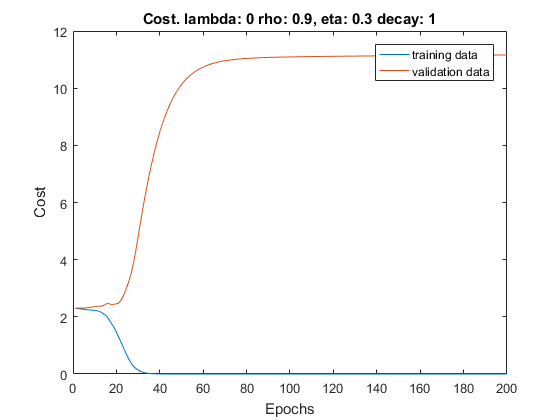


Figure 3. Overtraining with momentum,

without decay

**Coarse search for λ and η**

A coarse random search was performed for values of η between and , and values for λ between and . The network ran for 5 epochs and the best hyper-parameters can be seen below.

Table 2. Accuracy on the validation set for different

hyper-parameters when training for only 5 epochs.

|  |  |  |
| --- | --- | --- |
| η | λ | Accuracy |
| 0.041249 | 0.000000582 | 0.423200 |
| 0.029714 | 0.000507203 | 0.419600 |
| 0.055826 | 0.000000050 | 0.411700 |

**Fine search for λ and η**

In the fine search, the values of η searched were in between and , and values of λ were between and . The network ran for 7 epochs and the best hyper-parameters can be seen below.

Table 3. Accuracy on the validation set for different

hyper-parameters when training for 7 epochs.

|  |  |  |
| --- | --- | --- |
| η | λ | Accuracy |
| 0.030162 | 0.0002635966 | 0.434400 |
| 0.034456 | 0.0026442543 | 0.433400 |
| 0.024252 | 0.0000081840 | 0.432900 |

**Final training**

The best parameters found were and

The network was trained using these hyper-parameters for 30 epochs, using 19 000 training data samples and 1000 validation samples. The training and validation cost can be seen in figure 4, and the networks performance on the test data was 46.80 %. From figure 4, you can see that the network gets overfitted after about 10 epochs, which usually means that λ was too low.

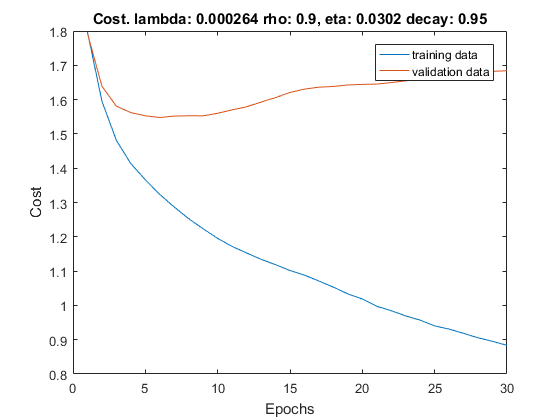


Figure 4. Training and validation cost when using 19000 training samples and 1000 validation samples.

Code

[X,Y,y] = LoadBatch('data\_batch\_1.mat');

subset = size(X,2);

featureSubset = size(X,1);

X = X(1:featureSubset,1:subset);

Y = Y(:,1:subset);

[val\_X,val\_Y,val\_y] = LoadBatch('data\_batch\_2.mat');

val\_X = val\_X(1:featureSubset,:);

[test\_X,test\_Y,test\_y] = LoadBatch('test\_batch.mat');

test\_X = test\_X(1:featureSubset,:);

% Subtract the mean of the training input

% on the training, validation and test input set

mean\_X = mean(X, 2);

X = X - repmat(mean\_X, [1, size(X, 2)]);

val\_X = val\_X - repmat(mean\_X, [1, size(val\_X, 2)]);

test\_X = test\_X - repmat(mean\_X, [1, size(test\_X, 2)]);

% For final testing with lots of data

% X = [X, val\_X(:, 1:9000)];

% Y = [Y, val\_Y(:, 1:9000)];

% y = [y, val\_y(:, 1:9000)];

%

% val\_X = val\_X(:, 9001:10000);

% val\_Y = val\_Y(:, 9001:10000);

% val\_y = val\_y(:, 9001:10000);

m = 50; % Number of hidden nodes

K = size(Y,1);

d = size(X,1);

N = size(X,2);

n\_epochs = 30;

n\_batch = 100;

[W,b] = InitializeParameters(d, K, m);

lambda = 0.000264;

eta = 0.0302;

decayRate = 0.95;

rho = 0.9;

[Wstar, bstar] = MiniBatchGD(X, Y, val\_X, val\_Y, val\_y, n\_batch, eta, n\_epochs, W, b, lambda, rho, decayRate);

%acc = ComputeAccuracy(val\_X, val\_y, Wstar, bstar);

%acc = ComputeAccuracy(test\_X, test\_y, Wstar, bstar)

%correct = CheckGradients(m)

%FindParameters(X, Y, val\_X, val\_Y, val\_y);

function correct = CheckGradients (m)

[X,Y,~] = LoadBatch('data\_batch\_1.mat');

N = 10;

d = 700;

K = size(Y,1);

X = X(1:d,1:N);

Y = Y(:,1:N);

global mean\_X;

mean\_X = mean(X, 2);

X = X - repmat(mean\_X, [1, size(X, 2)]);

[W,b] = InitializeParameters(d, K, m);

[s1, H, P] = EvaluateClassifier(X, W, b);

correct = 1;

lambda = 0;

[gradW, gradb] = ComputeGradients(X, H, s1, Y, P, W, lambda);

disp('Computed gradients');

%Checking gradients

[gradb\_num, gradW\_num] = ComputeGradsNumSlow(X, Y, W, b, lambda, 1e-5);

disp('W1 grad: ');

ga = gradW{1};

gn = gradW\_num{1};

relativeError = sqrt(sum(sum((ga - gn).^2))) / max(0.001, sum(sum(ga)) + sum(sum(gn)));

disp(['Relative error: ', num2str(relativeError)]);

maxDiff = max(max(abs(ga - gn)));

disp(['max difference: ', num2str(maxDiff)]);

if relativeError > 10E-4

correct = 0;

end

if maxDiff > 10E-6

correct = 0;

end

disp('W2 grad: ');

ga = gradW{2};

gn = gradW\_num{2};

relativeError = sqrt(sum(sum((ga - gn).^2))) / max(0.001, sum(sum(ga)) + sum(sum(gn)));

disp(['Relative error: ', num2str(relativeError)]);

maxDiff = max(max(abs(ga - gn)));

disp(['max difference: ', num2str(maxDiff)]);

if relativeError > 10E-4

correct = 0;

end

if maxDiff > 10E-6

correct = 0;

end

disp('b1 grad: ');

ga = gradb{1};

gn = gradb\_num{1};

relativeError = sqrt(sum(sum((ga - gn).^2))) / max(0.001, sum(sum(ga)) + sum(sum(gn)));

disp(['Relative error: ', num2str(relativeError)]);

maxDiff = max(max(abs(ga - gn)));

disp(['max difference: ', num2str(maxDiff)]);

if relativeError > 10E-4

correct = 0;

end

if maxDiff > 10E-6

correct = 0;

end

disp('b2 grad: ');

ga = gradb{2};

gn = gradb\_num{2};

relativeError = sqrt(sum(sum((ga - gn).^2))) / max(0.001, sum(sum(ga)) + sum(sum(gn)));

disp(['Relative error: ', num2str(relativeError)]);

maxDiff = max(max(abs(ga - gn)));

disp(['max difference: ', num2str(maxDiff)]);

if relativeError > 10E-4

correct = 0;

end

if maxDiff > 10E-6

correct = 0;

end

end

function acc = ComputeAccuracy(X, y, W, b)

%Calculate the accuracy scalar

% that is the percentage of correctly classified

% samples

[~, ~, P] = EvaluateClassifier(X, W, b);

sumCorrect = 0;

for sample=1:size(P,2)

[~, class] = max(P(:,sample));

if class == y(sample)

sumCorrect = sumCorrect + 1;

end

end

acc = sumCorrect / sample;

end

function J = ComputeCost(X, Y, W, b, lambda)

%Computes the cost

% J is a scalar with the sum of the loss of the network's

% predictions for the images in X relative

% to the labels and regularization term on W

s = 0;

[~, ~, P] = EvaluateClassifier(X, W, b);

N = size(X,2);

for i=1:N

cross = -log(dot(Y(:,i)',P(:,i)));

s = s + cross;

end

s = s / N;

J = s + lambda\*( sum(diag(W{1}.^2)) + sum(diag(W{2}.^2)));

end

function [gradW, gradb] = ComputeGradients(X, H, s1, Y, P, W, lambda)

%• each column of X corresponds to an image and it has size d×n.

%• each column of Y (K×n) is the one-hot ground truth label for the corresponding

% column of X.

%• each column of P contains the probability for each label for the image

% in the corresponding column of X. P has size K×n.

%• grad\_W1 has size m x d

%• grad\_W2 has size k x m

%• grad\_b1 has size m x 1

%• grad\_b2 has size k x 1

W1 = W{1};

W2 = W{2};

n = size(X,2);

m = size(W1,1);

k = size(W2,1);

gradW1 = zeros(size(W1));

gradW2 = zeros(size(W2));

gradb1 = zeros(m,1);

gradb2 = zeros(k,1);

for i=1:n

y = Y(:,i);

p = P(:,i);

x = X(:,i);

h = H(:,i);

s = s1(:,i);

g = - (y'/(y'\*p))\*(diag(p)-p\*p');

gradb2 = gradb2 + g';

gradW2 = gradW2 + g'\*h';

g = g\*W2;

ind = s > 0;

g = g\*diag(ind);

gradb1 = gradb1 + g';

gradW1 = gradW1 + g'\*x';

end

gradW1 = gradW1/n + 2\*lambda\*W1;

gradW2 = gradW2/n + 2\*lambda\*W2;

gradb1 = gradb1/n;

gradb2 = gradb2/n;

gradW = {gradW1, gradW2};

gradb = {gradb1, gradb2};

end

function [scores, H, P] = EvaluateClassifier(X, W, b)

%Evaluates the classifier by calculating the score

% and softmax

% each column of P contains the probability of each label

% for the image. P has size K\*N

W1 = W{1};

b1 = b{1};

W2 = W{2};

b2 = b{2};

M = size(W1,1);

K = size(W2,1);

N = size(X,2);

P = zeros(K,N);

scores = zeros(M, N);

for i=1:N

scores(:, i) = W1\*X(:,i) + b1;

end

H = max(scores, 0);

for i=1:N

s = W2\*H(:,i) + b2;

P(:,i) = exp(s)/dot(ones(K,1),exp(s));

end

end

function y = FindParameters(X, Y, val\_X, val\_Y, val\_y)

m = 50; % Number of hidden nodes

K = size(Y,1);

d = size(X,1);

n\_epochs = 10;

n\_batch = 100;

decayRate = 0.95;

rho = 0.9;

e\_min = -1.8;

e\_max = -1.3;

el\_min = -9;

el\_max = -2;

fileID = fopen('test.txt','a');

fprintf(fileID,'%8s\t%11s\t%8s\t%8s\n','eta', 'lambda', 'accuracy', 'average acc');

tries = 25;

el = el\_min + (el\_max - el\_min) \* rand(tries,1);

lambdas = 10.^el;

e = e\_min + (e\_max - e\_min) \* rand(tries,1);

etas = 10.^e;

for i=1:tries

bestAcc = 0;

averageAcc = 0;

iterations = 1;

for j=1:iterations

[W,b] = InitializeParameters(d, K, m);

lambda = lambdas(i,1);

eta = etas(i,1);

[Wstar, bstar] = MiniBatchGD(X, Y, val\_X, val\_Y, val\_y, n\_batch, eta, n\_epochs, W, b, lambda, rho, decayRate);

acc = ComputeAccuracy(val\_X, val\_y, Wstar, bstar);

if acc > bestAcc

bestAcc = acc;

end

averageAcc = averageAcc + acc;

end

averageAcc = averageAcc / iterations;

disp(['i: ', num2str(i)]);

A = [eta, lambda, bestAcc, averageAcc]

fprintf(fileID,'%0.6f\t%0.10f\t%0.6f\t%1.6f\n',A);

end

fclose(fileID);

y = 1;

end

function [W,b] = InitializeParameters(dim, numClasses, numHiddenNodes)

W1 = randn(numHiddenNodes,dim)\*0.001;

b1 = zeros(numHiddenNodes,1);

W2 = randn(numClasses,numHiddenNodes)\*0.001;

b2 = zeros(numClasses,1);

W = {W1, W2};

b = {b1, b2};

end

function [X, Y, y] = LoadBatch(filename)

%Function that reads the data from the file

% X is a matrix containing image pixel data.

% it has size d\*N, N is number of

% images = 10000, and d is dimensionality = 32\*32\*2=3072,

% each column represents one image

% Y contains on each column the one-hot represention of the label

% for each image

% and is the size N\*K where K is #labels = 10

% y is a row vector containing the label for each image, between 1 and 10

batch = load(filename);

X = double(batch.data')/255;

y = batch.labels' + 1;

N = size(X,2);

K = 10;

Y = zeros(K,N);

for i=1:N

Y(y(i),i) = 1;

end

end

function [Wstar, bstar] = MiniBatchGD(X, Y, Xval, Yval, yval, n\_batch, eta, n\_epochs, W, b, lambda, rho, decayRate)

%Mini-batch learning function of W and b, with gradient descent

% X training images

% Y labels for training images

% W and b initial values

% lambda regularization factor in the cost function

% GDparams contains n\_batch, eta and n\_epochs

N = size(X,2);

costTrain = zeros(1, n\_epochs);

costVal = zeros(1, n\_epochs);

mom\_W = {zeros(size(W{1})); zeros(size(W{2}))};

mom\_b = {zeros(size(b{1})); zeros(size(b{2}))};

decay = decayRate;

startEta = eta;

startCost = ComputeCost(X, Y, W, b, lambda);

for i=1:n\_epochs

for j=1:N/n\_batch

j\_start = (j-1)\*n\_batch + 1;

j\_end = j\*n\_batch;

Xbatch = X(:, j\_start:j\_end);

Ybatch = Y(:, j\_start:j\_end);

[s1, H, P] = EvaluateClassifier(Xbatch, W, b);

[grad\_W, grad\_b] = ComputeGradients(Xbatch, H, s1, Ybatch, P, W, lambda);

mom\_W{1} = mom\_W{1}\*rho + eta\*grad\_W{1};

W{1} = W{1} - mom\_W{1};

mom\_W{2} = mom\_W{2}\*rho + eta\*grad\_W{2};

W{2} = W{2} - mom\_W{2};

mom\_b{1} = mom\_b{1}\*rho + eta\*grad\_b{1};

b{1} = b{1} - mom\_b{1};

mom\_b{2} = mom\_b{2}\*rho + eta\*grad\_b{2};

b{2} = b{2} - mom\_b{2};

end

eta = eta \* decay;

costTrain(i) = ComputeCost(X, Y, W, b, lambda);

if costTrain(i)>3\*startCost

Wstar = W;

bstar = b;

disp(['Cost was to big: ', num2str(costTrain(i)), ' while start cost was: ', num2str(startCost)])

return

end

costVal(i) = ComputeCost(Xval, Yval, W, b, lambda);

disp(['epoch: ', num2str(i), '/', num2str(n\_epochs), ' Cost: ', num2str(costTrain(i))]);

end

Wstar = W;

bstar = b;

plot(1:n\_epochs, costTrain, 1:n\_epochs, costVal);

title(['Cost. lambda: ', num2str(lambda), ' rho: ', num2str(rho), ', eta: ', num2str(startEta), ' decay: ', num2str(decay)]);

xlabel('Epochs')

ylabel('Cost')

legend('training data', 'validation data')

acc = ComputeAccuracy(Xval, yval, W, b)

end